МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

**«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ Н.Г. ЧЕРНЫШЕВСКОГО»**

Кафедра теоретических основ компьютерной безопасности и криптографии

**Идеалы полугрупп**

ОТЧЁТ

ПО ДИСЦИПЛИНЕ

«ПРИКЛАДНАЯ УНИВЕРСАЛЬНАЯ АЛГЕБРА»

студента 3 курса 331 группы

специальности 10.05.01 Компьютерная безопасность

факультета компьютерных наук и информационных технологий

Алексеева Александра Александровича

|  |  |  |
| --- | --- | --- |
| Преподаватель  профессор, д.ф.-м.н. | \_\_\_\_\_\_\_\_\_\_ | В. А. Молчанов |
|  | подпись, дата |  |

Саратов 2022

**1 Постановка задачи**

Целью данной лабораторной работы является изучение строения полугрупп с помощью отношений Грина, а также разработка алгоритмов построения идеалов полугруппы по таблице Кэли, вычисление отношений Грина и построения «egg-box»-картины конечной полугруппы.

Порядок выполнения работы:

1. Рассмотреть понятия идеалов полугруппы. Разработать алгоритмы построения идеалов полугруппы по таблице Кэли.

2. Рассмотреть понятия и свойства отношений Грина на полугруппах.

3. Разработать алгоритмы вычисления отношений Грина и построения «egg-box»-картины конечной полугруппы.

2 Теория

2.1 Понятия идеалов полугруппы

Определение. Пусть – произвольная полугруппа. Непустое подмножество называется *правым* (соответственно, *левым*) *идеалом* *полугруппы* , если для любых выполняется условие: (соответственно ), т.е. (соответственно, ). Если – одновременно левый и правый идеал полугруппы , то называется *двустронним идеалом* (или просто *идеалом*) *полугруппы* . Ясно, что в коммутативной полугруппе все эти определения совпадают.

Лемма 1. Множество всех идеалов (соответственно, левых идеалов или правых идеалов ) любой полугруппы является системой замыкания.

Пусть – подмножество полугруппы . Тогда *наименьший правый* *идеал* полугруппы , содержащий подмножество , равен , *наименьший левый идеал* полугруппы , содержащий подмножество , равен и *наименьший идеал* полугруппы , содержащий подмножество , равен .

В частности, любой элемент определяет наименьшие правый, левый и двусторонний идеалы: , которые называются *главными* (соответственно, *правыми*, *левыми* и *двусторонними*) *идеалами*.

Минимальные относительно теоретико-множественного включения идеалы (соответственно, левые или правые идеалы) называются *минимальными идеалами* (соответственно, *минимальными левыми* или *правыми идеалами*).

2.2 Алгоритм построения идеалов полугруппы по таблице Кэли

*Вход*. Полугруппа *S*, размер полугруппы *N* и таблица Кэли *matrix*.

*Выход*. Множество правых идеалов *R*, множество левых идеалов *L* и множество двусторонних идеалов *I*.

Шаг 1. Строим все возможные подмножества множества *S*, т.е. создаётся множество *undersets*, которое содержит подмножества множества *S*: {*S*1}, {*S*2}, …, {*S*N}, {*S*1, *S*2}, {*S*1, *S*3}, …, {*S*1, *S*N}, {*S*2, *S*3}, …, {*S*2, *SN*­}, {*S*3, *S*4}, …, {*S*1, *S*2, …, *SN*}.

Шаг 2. Цикл по *i* от 1 до *size* (где *size* – размер множества *undersets*). Проверяем каждое множество *undersetsi* на принадлежность к правому, левому и двустороннему идеалу:

1) если ∀ *x* ∈ *undersetsi* и ∀ *y* ∈ *S* выполняется *matrix*[*x*][*y*] ∈ *undersetsi*, то *undersetsi* является правым идеалом ⇒ *undersetsi* добавляем в множество *R*;

2) если ∀ *x* ∈ *undersetsi* и ∀ *y* ∈ *S* выполняется *matrix*[*y*][*x*]∈ *undersetsi*, то *undersetsi* является левым идеалом ⇒ *undersetsi* добавляем в множество *L*;

3) если *undersetsi* является левым и правым идеалом, то *undersetsi* является двусторонним идеалом ⇒ *undersetsi* добавляем в множество *I*.

Шаг 3. Выводим множества *R*, *L* и *I*.

2.3 Понятия и свойства отношений Грина на полугруппах

Отображения  определяют ядра ℐ, ℛ ℒ по формулам:

ℐ ⟺

ℛ ⟺

ℒ ⟺

Все эти отношения, а также отношения 𝔇 = ℛ ∨ ℒ, ℋ = ℛ ∩ ℒ являются эквивалентностями на множестве , которые называются *отношениями Грина* полугруппы . Классы этих эквивалентностей, порожденные элементом , обозначаются , , , и , соответственно.

Лемма. Отношения Грина полугруппы S удовлетворяют следующим свойствам:

1) эквивалентность ℛ регулярна слева и эквивалентность ℒ регулярна справа, те. ℛ ⟹ ℛ и ℒ ⟹ ℒ для любых ,

2) эквивалентности ℛ, ℒ коммутируют,

3) 𝔇 = ℛ ∙ ℒ = ℒ ∙ ℛ,

4) если полугруппа конечна, то 𝔇 = ℐ,

5) любой класс эквивалентности 𝔇 можно изобразить с помощью следующей egg-box-диаграммы, клетки которой являются классами эквивалентности ℋ, лежащими в .

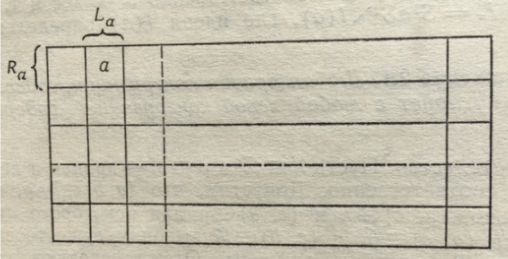


Рисунок 1 – egg-box-диаграмма

2.4 Алгоритм вычисления отношений Грина и построения «egg-box»-картины конечной полугруппы

*Вход.* Полугруппа *S*, её размер *N* и таблица Кэли *matrix*.

*Выход.* Отношения Грина *R*, *L*, *J*, *H*, *D* и «egg-box»-картины.

Шаг 1. Цикл по *i* от 1 до *N*. Для каждого *i* цикл по *j* от 1 до *N*.

Шаг 1.1. Строим (*Si*] и (*Sj*]. Если (*Si*] = (*Sj*], то пару (*Si*, *Sj*) добавляем в множество *R*.

Шаг 1.2. Строим [*Si*) и [*Sj*). Если [*Si*) = [*Sj*), то пару (*Si*, *Sj*) добавляем в множество *L*.

Шаг 1.3. Строим [*Si*] и [*Sj*]. Если [*Si*] = [*Sj*], то пару (*Si*, *Sj*) добавляем в множество *J*.

Шаг 2. Множество *H* = *R* ⋂ *L*.

Шаг 3. Множество *D* = *J*.

Шаг 4. Выводим отношения *R*, *L*, *J*, *H*, *D*.

Шаг 5. Находим классы эквивалентностей отношений *R*, *L*, *D*.

Шаг 6. Цикл по *n* от 1 до *size* (где *size* – количество классов эквивалентностей отношения *D*).

Шаг 6.1. Сохраняем классы эквивалентности отношения *R* ⊂ *Dn* в *rows*.

Шаг 6.2. Сохраняем классы эквивалентности отношения *L* ⊂ *Dn* в *columns*.

Шаг 6.3. Цикл по *i* от 1 до *sizeR* (*sizeR* – размер *rows*). Для каждого *i* цикл по *j* от 1 до *sizeC* (*sizeC* – размер *columns*). Выводим *rowsi* ⋂ *columnsj*.

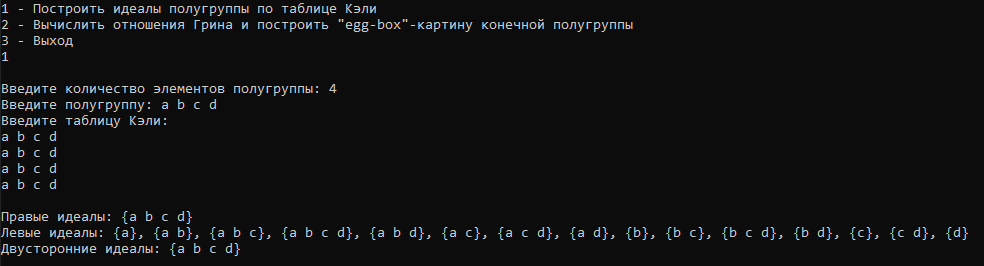
3 Результаты работы

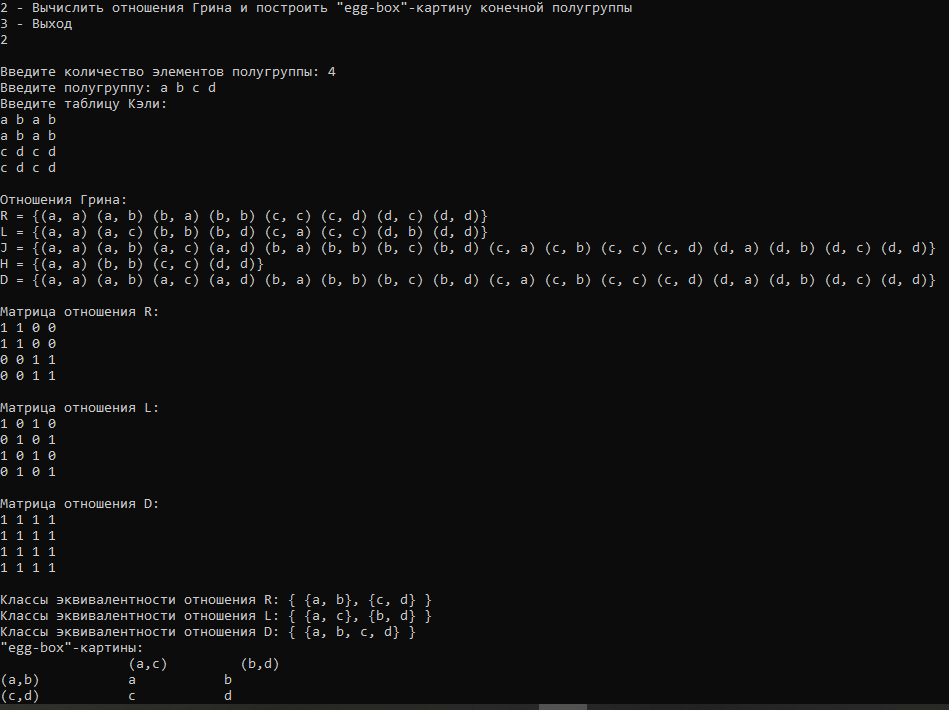
3.1 Оценка временной сложности алгоритмов

Алгоритм построения идеалов полугруппы по таблице Кэли – O();

Алгоритм вычисления отношений Грина и построения «egg-box»-картины конечной полугруппы – O(N3).

3.2 Результаты тестирования программы





3.3 Код программы

#include "iostream"

#include "vector"

#include "math.h"

#include "algorithm"

#include "set"

#include "string"

#include "map"

#include "iomanip"

using namespace std;

/////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

/\* Построить идеалы полугруппы по таблице Кэли \*/

/////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

int findNumber(char word, char\* el, int N) { //Поиск номера буквы в элементах полугруппы

for (int i = 0; i < N; i++)

if (el[i] == word)

return i;

}

bool findElUnderset(char word, set <char> underset, int size) { //Поиск елемента в подмножестве

for (char help : underset)

if (help == word)

return true;

return false;

}

bool associativity(char\*\* mas, int N, char\* halfgroup) { //Проверка свойства ассоциативности

for (int a = 0; a < N; a++)

for (int b = a; b < N; b++)

for (int c = 0; c < N; c++)

if (mas[findNumber(mas[a][b], halfgroup, N)][c] != mas[a][findNumber(mas[b][c], halfgroup, N)])

return false;

return true;

}

void makeUnderset(set<set <char>>& undersets, set <char> underset, int d, char\* halfgroup, int N) { //Строим подмножество

if (underset.size() == d) {

undersets.insert(underset);

return;

}

for (int i = 0; i < N; i++) {

set <char> newUnderset = underset;

if (newUnderset.find(halfgroup[i]) == newUnderset.end()) {

newUnderset.insert(halfgroup[i]);

makeUnderset(undersets, newUnderset, d, halfgroup, N);

}

}

}

void makeIdeals() { //Строим идеалы

cout << "\nВведите количество элементов полугруппы: ";

int N;

cin >> N;

cout << "Введите полугруппу: ";

char\* halfgroup = new char[N];

for (int i = 0; i < N; i++)

cin >> halfgroup[i];

cout << "Введите таблицу Кэли: \n";

char\*\* matrix = new char\* [N];

for (int i = 0; i < N; i++) {

matrix[i] = new char[N];

for (int j = 0; j < N; j++)

cin >> matrix[i][j];

}

if (!associativity(matrix, N, halfgroup)) {

cout << "\nНе выполняется ассоциативность полугруппы! \n";

return;

}

set<set <char>> undersets;

for (int d = 1; d <= N; d++) { //Строим все возможные подмножества

set <char> underset;

makeUnderset(undersets, underset, d, halfgroup, N);

}

vector <vector <char>> idealR;

vector <vector <char>> idealL;

vector <vector <char>> ideal;

for (auto s : undersets) {

bool flagR = true;

bool flagL = true;

for (char word : s) {

for (int i = 0; i < N; i++) {

if (!findElUnderset(matrix[findNumber(word, halfgroup, N)][i], s, s.size())) {

flagR = false;

}

if (!findElUnderset(matrix[i][findNumber(word, halfgroup, N)], s, s.size())) {

flagL = false;

}

if (!flagR && !flagL)

break;

}

if (!flagR && !flagL)

break;

}

if (flagR) {

vector <char> res;

for (char word : s)

res.push\_back(word);

idealR.push\_back(res);

}

if (flagL) {

vector <char> res;

for (char word : s)

res.push\_back(word);

idealL.push\_back(res);

if (flagR)

ideal.push\_back(res);

}

}

cout << "\nПравые идеалы: ";

for (int i = 0; i < idealR.size(); i++) {

cout << "{";

for (int j = 0; j < idealR[i].size(); j++) {

if (j == idealR[i].size() - 1)

cout << char(idealR[i][j]);

else

cout << char(idealR[i][j]) << " ";

}

if (i == idealR.size() - 1)

cout << "}";

else

cout << "}, ";

}

cout << "\nЛевые идеалы: ";

for (int i = 0; i < idealL.size(); i++) {

cout << "{";

for (int j = 0; j < idealL[i].size(); j++) {

if (j == idealL[i].size() - 1)

cout << char(idealL[i][j]);

else

cout << char(idealL[i][j]) << " ";

}

if (i == idealL.size() - 1)

cout << "}";

else

cout << "}, ";

}

cout << "\nДвусторонние идеалы: ";

for (int i = 0; i < ideal.size(); i++) {

cout << "{";

for (int j = 0; j < ideal[i].size(); j++) {

if (j == ideal[i].size() - 1)

cout << char(ideal[i][j]);

else

cout << char(ideal[i][j]) << " ";

}

if (i == ideal.size() - 1)

cout << "}";

else

cout << "}, ";

}

cout << endl;

}

/////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

/\* Вычислить отношения Грина и построить "egg-box"-картину конечной полугруппы \*/

/////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

set <char> srez(char\*\* matrix, int N, int num, string str) { //Срез по строке или столбцу матрицы

set <char> res;

if (str == "row")

for (int i = 0; i < N; i++)

res.insert(matrix[num][i]);

else

for (int i = 0; i < N; i++)

res.insert(matrix[i][num]);

return res;

}

bool entry(vector <char> A, vector <char> B) { //Проверяем, входит ли вектор В в вектор А

int count = 0;

for (int i = 0; i < A.size(); i++)

for (int j = 0; j < B.size(); j++)

if (A[i] == B[j])

count++;

if (count == B.size())

return true;

return false;

}

char generalEl(vector <char> A, vector <char> B) { //Находим общий элемент

for (int i = 0; i < A.size(); i++)

for (int j = 0; j < B.size(); j++)

if (A[i] == B[j])

return A[i];

}

void eggBox(char\*\* matrix, char\* halfgroup, int N, vector <pair <char, char>> R, vector <pair <char, char>> L, vector <pair <char, char>> D) { //Строим egg-box картину

//создаём матрицу отношения R

bool\*\* matrixR = new bool\*[N];

for (int i = 0; i < N; i++) {

matrixR[i] = new bool[N];

for (int j = 0; j < N; j++)

matrixR[i][j] = false;

}

for (int i = 0; i < R.size(); i++)

matrixR[findNumber(R[i].first, halfgroup, N)][findNumber(R[i].second, halfgroup, N)] = true;

cout << "\nМатрица отношения R: \n";

for (int i = 0; i < N; i++) {

for (int j = 0; j < N; j++)

cout << matrixR[i][j] << " ";

cout << endl;

}

//создаём матрицу отношения L

bool\*\* matrixL = new bool\* [N];

for (int i = 0; i < N; i++) {

matrixL[i] = new bool[N];

for (int j = 0; j < N; j++)

matrixL[i][j] = false;

}

for (int i = 0; i < L.size(); i++)

matrixL[findNumber(L[i].first, halfgroup, N)][findNumber(L[i].second, halfgroup, N)] = true;

cout << "\nМатрица отношения L: \n";

for (int i = 0; i < N; i++) {

for (int j = 0; j < N; j++)

cout << matrixL[i][j] << " ";

cout << endl;

}

//создаём матрицу отношения D

bool\*\* matrixD = new bool\* [N];

for (int i = 0; i < N; i++) {

matrixD[i] = new bool[N];

for (int j = 0; j < N; j++)

matrixD[i][j] = false;

}

for (int i = 0; i < D.size(); i++)

matrixD[findNumber(D[i].first, halfgroup, N)][findNumber(D[i].second, halfgroup, N)] = true;

cout << "\nМатрица отношения D: \n";

for (int i = 0; i < N; i++) {

for (int j = 0; j < N; j++)

cout << matrixD[i][j] << " ";

cout << endl;

}

//Вычисляем классы эквивалентности отношения R

vector <bool> used(N, false);

vector <vector <char>> classesR;

for (int i = 0; i < N; i++) {

vector <char> clas;

for (int j = 0; j < N; j++) {

if (matrixR[i][j] && !used[j]) {

clas.push\_back(halfgroup[j]);

used[j] = true;

}

}

if (!clas.empty())

classesR.push\_back(clas);

}

cout << "\nКлассы эквивалентности отношения R: { ";

int size = classesR.size();

for (int i = 0; i < size; i++) {

cout << "{";

int iSize = classesR[i].size();

for (int j = 0; j < iSize; j++) {

if (j == iSize - 1)

cout << classesR[i][j] << "}";

else

cout << classesR[i][j] << ", ";

}

if (i == size - 1)

cout << " } \n";

else

cout << ", ";

}

//Вычисляем классы эквивалентности отношения L

used.assign(N, false);

vector <vector <char>> classesL;

for (int i = 0; i < N; i++) {

vector <char> clas;

for (int j = 0; j < N; j++) {

if (matrixL[i][j] && !used[j]) {

clas.push\_back(halfgroup[j]);

used[j] = true;

}

}

if (!clas.empty())

classesL.push\_back(clas);

}

cout << "Классы эквивалентности отношения L: { ";

size = classesL.size();

for (int i = 0; i < size; i++) {

cout << "{";

int iSize = classesL[i].size();

for (int j = 0; j < iSize; j++) {

if (j == iSize - 1)

cout << classesL[i][j] << "}";

else

cout << classesL[i][j] << ", ";

}

if (i == size - 1)

cout << " } \n";

else

cout << ", ";

}

//Вычисляем классы эквивалентности отношения D

used.assign(N, false);

vector <vector <char>> classesD;

for (int i = 0; i < N; i++) {

vector <char> clas;

for (int j = 0; j < N; j++) {

if (matrixD[i][j] && !used[j]) {

clas.push\_back(halfgroup[j]);

used[j] = true;

}

}

if (!clas.empty())

classesD.push\_back(clas);

}

cout << "Классы эквивалентности отношения D: { ";

size = classesD.size();

for (int i = 0; i < size; i++) {

cout << "{";

int iSize = classesD[i].size();

for (int j = 0; j < iSize; j++) {

if (j == iSize - 1)

cout << classesD[i][j] << "}";

else

cout << classesD[i][j] << ", ";

}

if (i == size - 1)

cout << " } \n";

else

cout << ", ";

}

//Строим egg-box картину

cout << "\"egg-box\"-картины:";

for (int n = 0; n < classesD.size(); n++) {

vector <vector <char>> rows;

for (int i = 0; i < classesR.size(); i++)

if (entry(classesD[n], classesR[i]))

rows.push\_back(classesR[i]);

vector <vector <char>> columns;

for (int i = 0; i < classesL.size(); i++)

if (entry(classesD[n], classesL[i]))

columns.push\_back(classesL[i]);

//Выводим название столбцов

cout << "\n " << setw(10);

size = columns.size();

for (int i = 0; i < size; i++) {

cout << "(";

int iSize = columns[i].size();

for (int j = 0; j < iSize; j++) {

if (j == iSize - 1)

cout << columns[i][j] << ")" << setw(10);

else

cout << columns[i][j] << ",";

}

}

//Выводим название строк и сами строки

cout << " \n";

size = rows.size();

for (int i = 0; i < size; i++) {

cout << "(";

int iSize = rows[i].size();

for (int j = 0; j < iSize; j++) {

if (j == iSize - 1)

cout << rows[i][j] << ")" << setw(12);

else

cout << rows[i][j] << ",";

}

iSize = columns.size();

for (int j = 0; j < iSize; j++) {

cout << generalEl(rows[i], columns[j]) << setw(12);

}

cout << " \n";

}

}

}

void makeRelsGreen() { //Вычисляем отношения Грина

cout << "\nВведите количество элементов полугруппы: ";

int N;

cin >> N;

cout << "Введите полугруппу: ";

char\* halfgroup = new char[N];

for (int i = 0; i < N; i++)

cin >> halfgroup[i];

cout << "Введите таблицу Кэли: \n";

char\*\* matrix = new char\* [N];

for (int i = 0; i < N; i++) {

matrix[i] = new char[N];

for (int j = 0; j < N; j++)

cin >> matrix[i][j];

}

if (!associativity(matrix, N, halfgroup)) {

cout << "\nНе выполняется ассоциативность полугруппы! \n";

return;

}

vector <pair <char, char>> R;

vector <pair <char, char>> L;

vector <pair <char, char>> J;

vector <pair <char, char>> H;

for (int i = 0; i < N; i++)

for (int j = 0; j < N; j++) {

set <char> a = srez(matrix, N, i, "row");

set <char> b = srez(matrix, N, j, "row");

if (a == b)

R.push\_back(make\_pair(halfgroup[i], halfgroup[j]));

a = srez(matrix, N, i, "column");

b = srez(matrix, N, j, "column");

if (a == b)

L.push\_back(make\_pair(halfgroup[i], halfgroup[j]));

set <char> resA;

a = srez(matrix, N, i, "column");

for (char word : a) {

set <char> help;

help = srez(matrix, N, findNumber(word, halfgroup, N), "row");

for (char ans : help)

resA.insert(ans);

}

set <char> resB;

b = srez(matrix, N, j, "column");

for (char word : b) {

set <char> help;

help = srez(matrix, N, findNumber(word, halfgroup, N), "row");

for (char ans : help)

resB.insert(ans);

}

if (resA == resB)

J.push\_back(make\_pair(halfgroup[i], halfgroup[j]));

}

for (int i = 0; i < R.size(); i++)

for (int j = 0; j < L.size(); j++)

if (R[i] == L[j])

H.push\_back(R[i]);

cout << "\nОтношения Грина: \nR = {";

int size = R.size();

for (int i = 0; i < size; i++) {

if (i == size - 1)

cout << "(" << R[i].first << ", " << R[i].second << ")} \n";

else

cout << "(" << R[i].first << ", " << R[i].second << ") ";

}

cout << "L = {";

size = L.size();

for (int i = 0; i < size; i++) {

if (i == size - 1)

cout << "(" << L[i].first << ", " << L[i].second << ")} \n";

else

cout << "(" << L[i].first << ", " << L[i].second << ") ";

}

cout << "J = {";

size = J.size();

for (int i = 0; i < size; i++) {

if (i == size - 1)

cout << "(" << J[i].first << ", " << J[i].second << ")} \n";

else

cout << "(" << J[i].first << ", " << J[i].second << ") ";

}

cout << "H = {";

size = H.size();

for (int i = 0; i < size; i++) {

if (i == size - 1)

cout << "(" << H[i].first << ", " << H[i].second << ")} \n";

else

cout << "(" << H[i].first << ", " << H[i].second << ") ";

}

cout << "D = {";

size = J.size();

for (int i = 0; i < size; i++) {

if (i == size - 1)

cout << "(" << J[i].first << ", " << J[i].second << ")} \n";

else

cout << "(" << J[i].first << ", " << J[i].second << ") ";

}

eggBox(matrix, halfgroup, N, R, L, J);

}

/////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

/\* Главная функция \*/

/////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

int main() {

setlocale(LC\_ALL, "ru");

for (;;) {

cout << "1 - Построить идеалы полугруппы по таблице Кэли \n2 - Вычислить отношения Грина и построить \"egg-box\"-картину конечной полугруппы \n3 - Выход \n";

int x;

cin >> x;

switch (x) {

case 1:

makeIdeals();

cout << endl;

break;

case 2:

makeRelsGreen();

cout << endl;

break;

case 3:

return 0;

default:

cout << "Incorrect. Try again! \n";

}

}

}

ЗАКЛЮЧЕНИЕ

В ходе данной работы были изучены и реализованы алгоритмы построения идеалов полугруппы по таблице Кэли, вычисления отношений Грина и построения «egg-box»-картины. Сложность первого из них зависит от размера множества всевозможных неповторяющихся подмножеств, составленных из элементов полугруппы, размера каждого из этих подмножеств и размера самой полугруппы. Сложность второго алгоритма зависит только от количества элементов в полугруппе.